TÉCNICO LISBOA

Fundamentos da Programação

Projeto 2

Hello Quantum

2018

1 Trabalho a Realizar

O jogo *Hello Quantum* foi descrito no primeiro projeto. O objetivo deste segundo projeto é escrever um programa em Python que permita jogar um jogo de *Hello Quantum*.

Para tal, deverá definir um conjunto de tipos abstratos de dados que deverão ser utilizados para manipular a informação necessária ao decorrer do jogo, bem como um conjunto de funções adicionais que permitirão jogar o jogo propriamente dito.

Deverá ainda considerar as operações associadas às portas descritas no primeiro projeto, e que se transcrevem em baixo.

1.1 Tipos Abstratos de Dados

NOTA: Apenas os construtores e as funções para as quais a verificação da correção dos argumentos é explicitamente pedida devem verificar a validade dos argumentos. Apenas os modificadores alteram de modo destrutivo o seu argumento. Todas as funções de alto nível (ou seja que não correspondem a operações básicas) devem respeitar as barreiras de abstração.

1.1.1 Celula (1,5 val.)

O tipo celula é usado para representar a cor de cada um dos círculos de um qubit, podendo assumir três valores, um por cada estado possível: `ativo`, `inativo` e `incerto`. As operações básicas associadas a este tipo são:

- Construtor
  - *cria_celula*: $\{1, 0, -1\} \mapsto$ *celula*
    cria_celula(v), recebe o valor do estado de uma célula do qubit (`0`, representa *inativo*, `1`, *ativo* e `-1` *incerto*) e devolve uma celula com esse valor. Verifica a validade do seu argumento, gerando um `ValueError` com a mensagem

`'cria_celula: argumento invalido'` caso o seu argumento não seja válido.

- Seletor
  - *obter_valor*: *celula* $\mapsto$ *{1, 0, -1}*
    obter_valor(c) devolve o valor associado a *c*.
- Modificador
  - *inverte_estado*: *celula* $\mapsto$ *celula*
    inverte(c) devolve a celula resultante de inverter o estado da celula que é seu argumento: uma célula *inativa* torna-se *ativa*; uma célula *ativa* torna-se *inativa* e uma célula no estado *incerto* mantém-se no mesmo estado.
- Reconhecedor
  - *eh_celula*: *universal* $\mapsto$ *lógico*
    eh_celula(arg) devolve *verdadeiro* apenas no caso do seu argumento ser do tipo celula.
- Teste
  - *celulas_iguais*: *celula*^{2} $\mapsto$ *lógico*
    celulas_iguais(c_1, c_2) devolve *verdadeiro* apenas se c_1 e c_2 são células no mesmo estado.
- Transformador
  - *celula_para_str*: *celula* $\mapsto$ *cad. caracteres*
    celula_para_str(c) devolve uma cadeia de caracteres que representa a celula que é seu argumento: o estado *ativo* é representado por `'1'`; o estado *inativo* é representado por `'0'` e o estado *incerto* é representado por `'x'`.

Exemplo de interação:

```
>>> c0 = cria_celula(0)
>>> c1 = cria_celula(1)
>>> cx = cria_celula(-1)
>>> celula_para_str(c1)
'1'
>>> celula_para_str(c0)
'0'
>>> celula_para_str(cx)
'x'
>>> celulas_iguais(c1,c0)
False
>>> c2 = cria_celula(1)
>>> celulas_iguais(c1,c2)
True
>>> eh_celula(c0)
True
```

```
>>> eh_celula(-2)
False
>>> eh_celula(2)
False
>>> celula_para_str(inverte_estado(c0))
'1'
>>> celula_para_str(inverte_estado(c1))
'0'
>>> celula_para_str(inverte_estado(cx))
'x'
```

1.1.2 Coordenada (1,5 val.)

O tipo coordenada é utilizado para representar a posição de uma célula num conjunto de células, organizadas numa estrutura semelhante a uma matriz de três linhas por três colunas. As operações básicas associadas a este tipo são:

- Construtor
  - *cria_coordenada*: $\mathbb{N}^2 \mapsto$ *coordenada*
    cria_coordenada(l, c) devolve a coordenada correspondente à linha l e à coluna c. Caso os argumentos não sejam números naturais ou um dos argumentos não pertença ao conjunto $\{0, 1, 2\}$, a função é indefinida, na implementação devolve um `ValueError` com a mensagem `'cria_coordenada: argumentos invalidos'`.
- Selectores
  - *coordenada_linha*: *coordenada* $\mapsto \mathbb{N}$
    coordenada_linha(c) devolve o natural correspondente à linha da coordenada *c*.
  - *coordenada_coluna*: *coordenada* $\mapsto \mathbb{N}$
    coordenada_coluna(c) devolve o natural correspondente à coluna da coordenada *c*.
- Reconhecedor
  - *eh_coordenada*: *universal* $\mapsto$ *lógico*
    eh_coordenada(arg) devolve *verdadeiro* apenas no caso em que o seu argumento é do tipo coordenada.
- Teste
  - *coordenadas_iguais*: *coordenada* $\times$ *coordenada* $\mapsto$ *lógico*
    coordenadas_iguais(c_1, c_2) devolve *verdadeiro* apenas se c_1 e c_2 representem coordenadas iguais, ou seja representam a mesma posição.
- Transformador
  - *coordenada_para_str*: *coordenada* $\mapsto$ *cad. caracteres*
    coordenada_para_str(c) devolve a cadeia de caracteres que representa o seu argumento. A coordenada correspondente à linha *l* e coluna *c* é representada pela cadeia de carateres '(l, c)'.

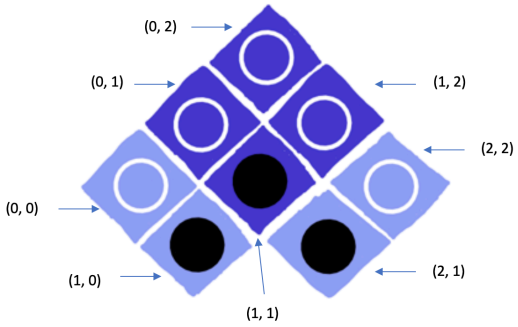

Figure 1: Tabuleiro.

Exemplo de interação:

```
>>> cd = cria_coordenada(0, 2)
>>> c = cria_coordenada(0, 'a')
Traceback (most recent call last):...
builtins.ValueError: cria_coordenada: argumentos invalidos.
>>> coordenada_linha(cd)
0
>>> coordenada_coluna(cd)
2
>>> eh_coordenada(cd)
True
>>> eh_coordenada(('a',0))
False
>>> coordenada_para_str(cd)
'(0, 2)'
>>> coordenadas_iguais(cd, cria_coordenada(0,2))
True
>>> coordenadas_iguais(cd, cria_coordenada(2,0))
False
```

1.1.3 Tabuleiro (3,0 val.)

O tipo tabuleiro é utilizado para guardar um par de qubits e as células de observação para questionar os seus valores. Um tabuleiro corresponde a uma matriz com 3 linhas e 3 colunas como representado na Figura 1. As operações básicas associadas ao tipo tabuleiro são:

- Construtores
  - *tabuleiro_inicial*: $\{\} \mapsto$ *tabuleiro*
    tabuleiro_inicial() devolve o tabuleiro que representa o seu estado inicial do jogo, ou seja, o representado na Figura 1. Note-se que a coordenada $(2, 0)$ não existe.

  - *str_para_tabuleiro*: *cad. caracteres* $\mapsto$ *tabuleiro*
    str_para_tabuleiro(s) devolve o tabuleiro correspondente à cadeia de caracteres que é seu argumento. A cadeia de caracteres s corresponde à representação interna do tabuleiro tal como usada no primeiro projeto, ou seja, uma cadeia de caracteres correspondente a um tuplo de 3 tuplos, os dois primeiros com 3 elementos e o último com 2 elementos. Os elementos destes tuplos são obtidos do conjunto $\{0, 1, -1\}$. Por exemplo, `'((-1, -1, -1), (0, 1, -1), (1, -1))'`. A sua função deverá garantir a correção do argumento, gerando um `ValueError` com a mensagem `'str_para_tabuleiro: argumentos invalidos'`.

- Seletores
  - *tabuleiro_dimensao*: *tabuleiro* $\mapsto \mathbb{N}$
    tabuleiro_dimensao(t) devolve o natural correspondente ao número de linhas (e, consequentemente, também ao número de colunas) existentes em *t*.
  - *tabuleiro_celula*: *tabuleiro* $\times$ *coordenada* $\mapsto$ *celula*
    tabuleiro_celula(t, coor) devolve a celula presente na coordenada *coor* do tabuleiro *t*.

- Modificadores
  - *tabuleiro_substitui_celula*: *tabuleiro* $\times$ *celula* $\times$ *coordenada* $\rightarrow$ *tabuleiro*
    tabuleiro_substitui_celula(t, cel, coor) devolve o tabuleiro que resulta de substituir a célula existente na coordenada *coor* do tabuleiro, pela nova célula. A sua função deve verificar a correção dos argumentos, gerando um `ValueError` com a mensagem `'tabuleiro_substitui_celula: argumentos invalidos'`.
  - *tabuleiro_inverte_estado*: *tabuleiro* $\times$ *coordenada* $\rightarrow$ *tabuleiro*
    tabuleiro_inverte_estado(t, coor) devolve o tabuleiro que resulta de inverter o estado da célula presente na coordenada *coor* do tabuleiro. A sua função deve verificar a correção dos argumentos, gerando um `ValueError` com a mensagem `'tabuleiro_inverte_estado: argumentos invalidos'`.

- Reconhecedor
  - *eh_tabuleiro*: *universal* $\rightarrow$ *lógico*
    eh_tabuleiro(arg) devolve *verdadeiro* apenas no caso de *arg* ser do tipo tabuleiro.

- Teste
  - *tabuleiros_iguais*: *tabuleiro* $\times$ *tabuleiro* $\rightarrow$ *lógico*
    tabuleiros_iguais(t_1, t_2) devolve *verdadeiro* apenas no caso de t_1 e t_2 forem tabuleiros que contenham células iguais em cada uma das coordenadas.

- Transformador
  - *tabuleiro_para_str*: *tabuleiro* → *cad. caracteres*
    tabuleiro_para_str(t) devolve a cadeia de caracteres que represente o seu argumento. A representação externa é idêntica à apresentada no primeiro projeto, e de acordo com o exemplo na seguinte interação (note que o tabuleiro é representado com os qubits fazendo um ângulo de 45 graus com a horizontal).

Exemplo de interação:

```
>>> t0 = tabuleiro_inicial()
>>> print(tabuleiro_para_str(t0))
+-------+
|...x...|
|..x.x..|
|.x.0.x.|
|..0.0..|
+-------+
>>> t2 = str_para_tabuleiro('((-1, -1, -1), (0, 1, -1), (1, -1))')
>>> print(tabuleiro_para_str(t2))
+-------+
|...x...|
|..x.x..|
|.x.1.x.|
|..0.1..|
+-------+
>>> celula_para_str(tabuleiro_celula(t0, cria_coordenada(0,0)))
'x'
>>> celula_para_str(tabuleiro_celula(t0, cria_coordenada(1,1)))
'0'
>>> eh_tabuleiro(t0)
True
>>> t1 = tabuleiro_inverte_estado(t0, cria_coordenada(1,1))
>>> print(tabuleiro_para_str(t1))
+-------+
|...x...|
|..x.x..|
|.x.1.x.|
|..0.0..|
+-------+
>>> tabuleiros_iguais(t0, tabuleiro_inicial())
False
>>> tabuleiros_iguais(t0, t1)
True
```

As operações de alto nível associadas ao tipo tabuleiro são:

- *porta_x*: *tabuleiro* × *{'E', 'D'}* ↦ *tabuleiro* **(1 val.)**

  porta_x(t, p) devolve o tabuleiro resultante de aplicar a porta X à célula inferior do qubit da esquerda, ou da direita, conforme *p* seja 'E' ou 'D', respetivamente. A função deve verificar a validade dos seus argumentos, gerando um `ValueError` com a mensagem `'porta_x: argumentos invalidos.'` caso os argumentos não sejam válidos.

  Exemplo:

```
>>> t1 = porta_x(tabuleiro_inicial(), 'E')
>>> print(tabuleiro_para_str(t1))
+-------+
|...x...|
|..x.x..|
|.x.1.x.|
|..1.0..|
+-------+
>>> t2 = porta_x(t1,'D')
>>> print(tabuleiro_para_str(t2))
+-------+
|...x...|
|..x.x..|
|.x.0.x.|
|..1.1..|
+-------+
>>> p = porta_x(t2, 'X')
Traceback (most recent call last): <...>
builtins.ValueError: porta_x: argumentos invalidos.
```

- *porta_z*: *tabuleiro* × *{'E', 'D'}* ↦ *tabuleiro* **(1 val.)**

  porta_z(t, p) devolve o tabuleiro resultante de aplicar a porta Z à célula superior do qubit da esquerda, ou da direita, conforme *p* seja 'E' ou 'D', respetivamente. A função deve verificar a validade dos seus argumentos, gerando um `ValueError` com a mensagem `'porta_z: argumentos invalidos.'` caso os argumentos não sejam válidos.

  Exemplo:

```
>>> t3 = str_para_tabuleiro('((0, -1, 0), (-1, -1, -1), (-1, 0))')
>>> print(tabuleiro_para_str(t3))
+-------+
|...0...|
|..x.x..|
|.0.x.0.|
|..x.x..|
+-------+
>>> t4 = porta_z(t3, 'E')
```

```
>>> print(tabuleiro_para_str(t4))
+-------+
|...1...|
|..x.x..|
|.1.x.0.|
|..x.x..|
+-------+
>>> t5 = porta_z(t4, 'D')
print(tabuleiro_para_str(t5))
+-------+
|...0...|
|..x.x..|
|.1.x.1.|
|..x.x..|
+-------+
```

- *porta_h*: *tabuleiro* $\times$ *{'E', 'D'}* $\mapsto$ *tabuleiro* **(1 val.)**

  porta_h(t, p) devolve o tabuleiro resultante de aplicar a porta H ao qubit da esquerda, ou da direita, conforme *p* seja 'E' ou 'D', respetivamente. A função deve verificar a validade dos seus argumentos, gerando um `ValueError` com a mensagem `'porta_h: argumentos invalidos.'` caso os argumentos introduzidos não sejam válidos.

  Exemplo:

```
>>> t1 = tabuleiro_inicial()
>>> print(tabuleiro_para_str(t1))
+-------+
|...x...|
|..x.x..|
|.x.0.x.|
|..0.0..|
+-------+
>>> t2 = porta_h(t1,'D')
>>> print(tabuleiro_para_str(t2))
+-------+
|...x...|
|..x.0..|
|.x.x.0.|
|..0.x..|
+-------+
>>> t3 = porta_h(t2,'E')
>>> print(tabuleiro_para_str(t3))
+-------+
|...0...|
|..x.x..|
|.0.x.0.|
|..x.x..|
+-------+
```

```
>>> t = porta_h(t2,'X')
Traceback (most recent call last):<...>
builtins.ValueError: porta_h: argumentos invalidos.
```

1.2 Funções adicionais

- *hello_quantum*: *cad. caracteres* $\mapsto$ *lógico* **(3 val.)**

  Função principal do jogo que permite jogar um jogo completo de $Hello\ Quantum$. A função *hello_quantum* recebe uma cadeia de caracteres contendo a descrição do tabuleiro objetivo e do número máximo de jogadas. A função devolve *verdadeiro* se o jogador conseguir transformar o tabuleiro inicial no tabuleiro objetivo, não ultrapassando o número de jogadas indicado e devolve *falso* em caso contrário. O número máximo de jogadas é apresentado imediatamente após a descrição do tabuleiro objetivo, separado por ':'. Por exemplo, `hello_quantum('((-1, -1, -1), (0, 1, -1), (1, -1)):1')`.

  Exemplo 1:

```
>>> hello_quantum('((-1, -1, -1), (0, 1, -1), (1, -1)):1')
Bem-vindo ao Hello Quantum!
O seu objetivo e chegar ao tabuleiro:
+-------+
|...x...|
|..x.x..|
|.x.1.x.|
|..0.1..|
+-------+
Comecando com o tabuleiro que se segue:
+-------+
|...x...|
|..x.x..|
|.x.0.x.|
|..0.0..|
+-------+
Escolha uma porta para aplicar (X, Z ou H): X
Escolha um qubit para analisar (E ou D): D
+-------+
|...x...|
|..x.x..|
|.x.1.x.|
|..0.1..|
+-------+
Parabens, conseguiu converter o tabuleiro em 1 jogadas!
True
```

  Exemplo 2:

```
>>> hello_quantum('((1, -1, 0), (-1, -1, -1), (-1, 1)):4')
Bem-vindo ao Hello Quantum!
O seu objetivo e chegar ao tabuleiro:
```

```
+-------+
|...0...|
|..x.x..|
|.1.x.1.|
|..x.x..|
+-------+
Comecando com o tabuleiro que se segue:
+-------+
|...x...|
|..x.x..|
|.x.0.x.|
|..0.0..|
+-------+
Escolha uma porta para aplicar (X, Z ou H): H
Escolha um qubit para analisar (E ou D): D
+-------+
|...x...|
|..x.0..|
|.x.x.0.|
|..0.x..|
+-------+
Escolha uma porta para aplicar (X, Z ou H): Z
Escolha um qubit para analisar (E ou D): D
+-------+
|...x...|
|..x.1..|
|.x.x.1.|
|..0.x..|
+-------+
Escolha uma porta para aplicar (X, Z ou H): H
Escolha um qubit para analisar (E ou D): E
+-------+
|...1...|
|..x.x..|
|.0.x.1.|
|..x.x..|
+-------+
Escolha uma porta para aplicar (X, Z ou H): Z
Escolha um qubit para analisar (E ou D): E
+-------+
|...0...|
|..x.x..|
|.1.x.1.|
|..x.x..|
+-------+
Parabens, conseguiu converter o tabuleiro em 4 jogadas!
True
```

1.3 Sugestões

1. Leia o enunciado completo, procurando perceber o objetivo das várias funções pedidas. Em caso de dúvida de interpretação, utilize o horário de dúvidas para esclarecer as suas questões.

2. No processo de desenvolvimento do projeto, comece por implementar os vários tipos de dados e as várias funções pela ordem apresentada no enunciado, seguindo as metodologias estudadas na disciplina. Ao desenvolver cada um dos tipos de dados e as funções associadas, comece por pensar se pode usar algum(a) do(a)s anteriores.

3. Para verificar a funcionalidade das suas funções, utilize os exemplos fornecidos como casos de teste.

4. Tenha o cuidado de reproduzir fielmente as mensagens de erro e restantes outputs, conforme ilustrado nos vários exemplos.

2 Aspetos a evitar

Os seguintes aspetos correspondem a sugestões para evitar maus hábitos de trabalho (e, consequentemente, más notas no projeto):

1. Não pense que o projeto se pode fazer nos últimos dias. Se apenas iniciar o seu trabalho neste período irá ver a Lei de Murphy em funcionamento (todos os problemas são mais difíceis do que parecem; tudo demora mais tempo do que nós pensamos; e se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis).

2. Não duplique código. Se duas funções são muito semelhantes é natural que estas possam ser fundidas numa única, eventualmente com mais argumentos.

3. Não se esqueça que as funções excessivamente grandes são penalizadas no que respeita ao estilo de programação.

4. A atitude "vou pôr agora o programa a correr de qualquer maneira e depois preocupo-me com o estilo" é totalmente errada.

5. Quando o programa gerar um erro, preocupe-se em descobrir qual a causa do erro. As "marteladas" no código têm o efeito de distorcer cada vez mais o código.

3 Classificação

A avaliação da execução será feita através do sistema Mooshak, onde, tal como no primeiro projeto, existem vários testes configurados no sistema. O tempo de execução de cada teste está limitado, bem como a memória utilizada. O sistema não deverá ser utilizado para debug e como tal, só poderá efetuar uma nova submissão pelo menos 15 minutos depois da submissão anterior. Só são permitidas 10 submissões em simultâeo no sistema, pelo

que uma submissão poderá ser recusada se este limite for excedido.[1] Nesse caso tente mais tarde.

Os testes considerados para efeitos de avaliação podem incluir ou não os exemplos disponibilizados, além de um conjunto de testes adicionais. O facto de um projeto completar com sucesso os exemplos fornecidos não implica, pois, que esse projeto esteja totalmente correto, pois o conjunto de exemplos fornecido não é exaustivo. É da responsabilidade de cada aluno garantir que o código produzido está correto.

Não será disponibilizado qualquer tipo de informação sobre os casos de teste utilizados pelo sistema de avaliação automática. Os ficheiros de teste usados na avaliação do projeto serão disponibilizados na página da disciplina após a data de entrega.

A nota do projeto será baseada nos seguintes aspetos:

1. Execução correta (**12 valores**). Esta parte da avaliação é feita recorrendo ao Mooshak que define uma nota face aos vários aspetos considerados.
2. Respeito pelas barreiras de abstração (**4 valores**). Esta parte da avaliação é feita recorrendo ao Mooshak que define uma nota face aos vários aspetos considerados.
3. Estilo de programação e facilidade de leitura, nomeadamente a abstração procedimental, a abstração de dados, nomes bem escolhidos, qualidade (e não quantidade) dos comentários e tamanho das funções (**4 valores**). Os seus comentários deverão incluir, entre outros, a assinatura de cada função definida assim como uma descrição da representação interna adotada em cada um dos tipos de dados definidos.

4 Condições de realização e prazos

A entrega do 2º projeto será efetuada exclusivamente por via eletronica. Deverá submeter o seu projeto através do sistema Mooshak, até às **23:59 do dia 6 de Dezembro de 2018**. Depois desta hora, não serão aceites projetos sob pretexto algum.

Deverá submeter um único ficheiro com extensão `.py` contendo todo o código do seu projeto. O ficheiro de código deve conter em comentário, na primeira linha, o número e o nome do aluno.

No seu ficheiro de código **não podem** ser utilizados caracteres acentuados ou qualquer carácter que não pertença à tabela ASCII. Isto inclui comentários e cadeias de caracteres. Programas que não cumpram este requisito serão penalizados em três valores.

Pode ou não haver uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

Lembre-se que no Técnico, a fraude académica é levada muito a sério e que a cópia numa prova (projetos incluídos) leva à reprovação na disciplina. O corpo docente da cadeira será o único juiz do que se considera ou não copiar num projeto.

[1] Note que o limite de 10 submissões simultâneas no sistema Mooshak implica que, caso haja um número elevado de tentativas de submissão sobre o prazo de entrega, alguns grupos poderão não conseguir submeter nessa altura e verem-se, por isso, impossibilitados de submeter o código dentro do prazo.